ETH zürich

Runtime Verification with TeSSLa on Enzian

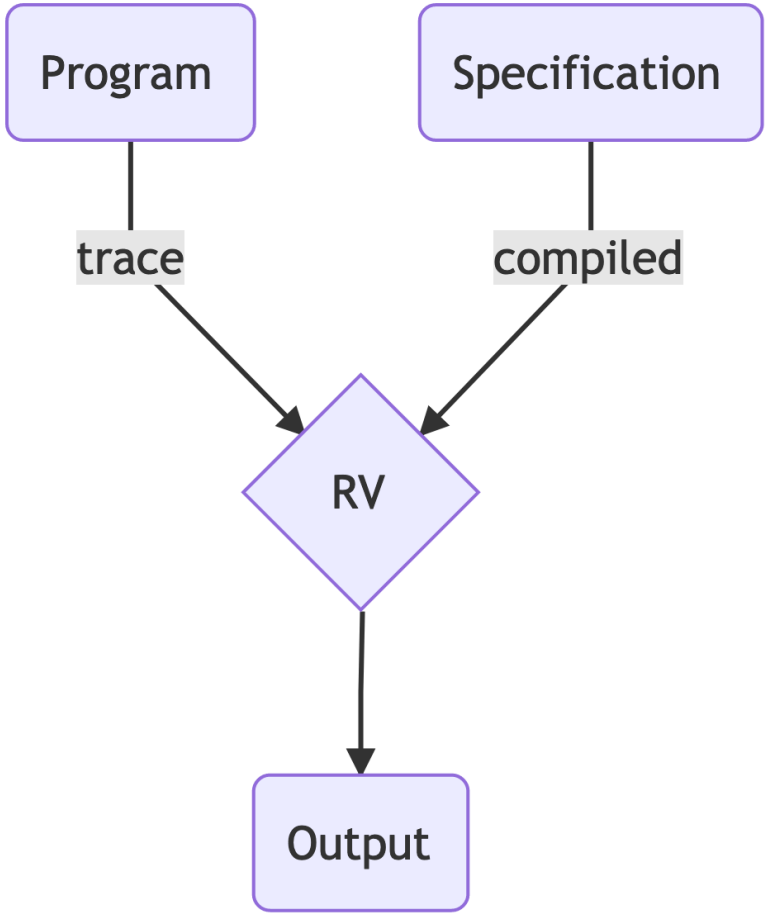
MSc Thesis Pirmin Schmid
Supervised by Prof. Timothy Roscoe and Dr. David Cock

ENZIAN

inf | Informatik Computer Science

Systems@ETH Zürich

Runtime Verification (RV)



Overview

- TeSSLa 101
- System overview & data flow
  - Design and implementation
  - Instrumentation
  - Evaluation
- Runtime verification examples
  - Memory allocation
  - Event handler: queueing time
  - Critical section: protected by lock
- Upgrade path to Enzian

TeSSLa 101

TeSSLa: Temporal Stream-based Specification Language*

Lukas Convent, Sebastian Hungerecker, Martin Leucker,
Torben Scheffel, Malte Schmitz, and Daniel Thoma

Institute for Software Engineering, University of Lübeck, Lübeck, Germany

TeSSLa 101: count() macro

TeSSLa: Temporal Stream-based Specification Language*

Lukas Convent, Sebastian Hungerecker, Martin Leucker,
Torben Scheffel, Malte Schmitz, and Daniel Thoma

Institute for Software Engineering, University of Lübeck, Lübeck, Germany

Event tuple: (timestamp, data value)

```
in event: Events[Int]




def event_count = count(event)

out event_count
```

macro count() from TeSSLa stdlib

TeSSLa 101: count() macro

```
in event: Events[Int]

def count[A](a: Events[A]) = c where {
    def c: Events[Int] = merge(last(c, a) + 1, 0)
}

def event_count = count(event)

out event_count
```

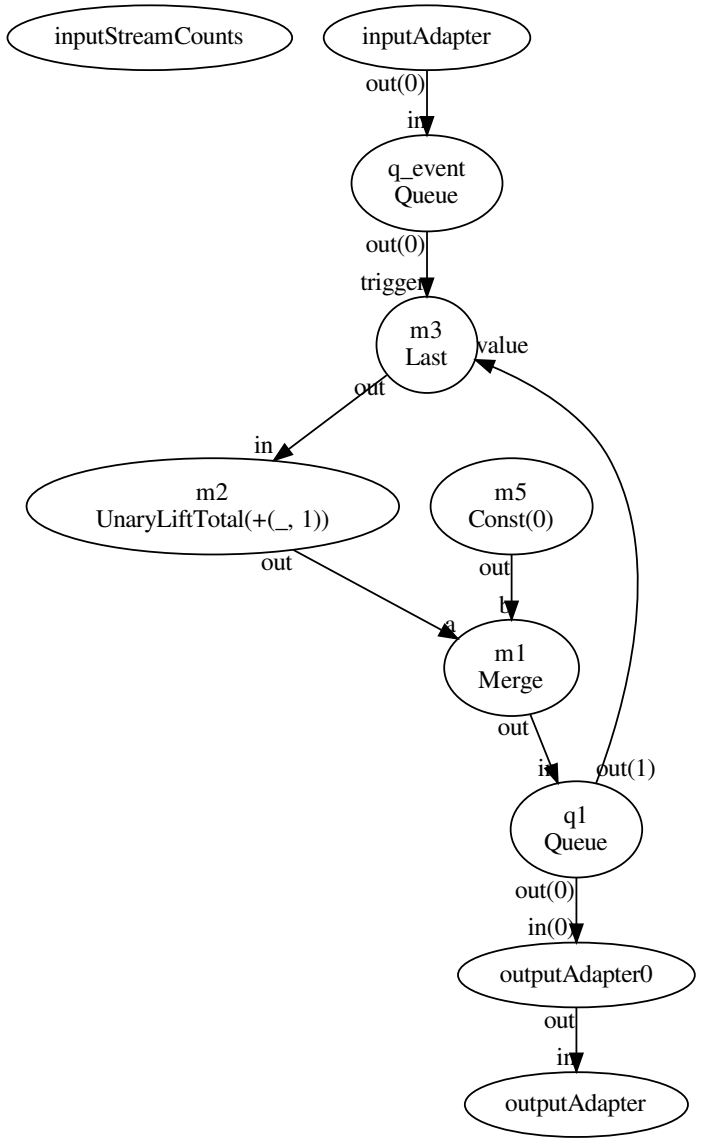
Event tuple: (timestamp, data value)

```
last(value_stream, trigger_stream)

merge(stream_a, stream_b)
```

macro count() from TeSSLa stdlib

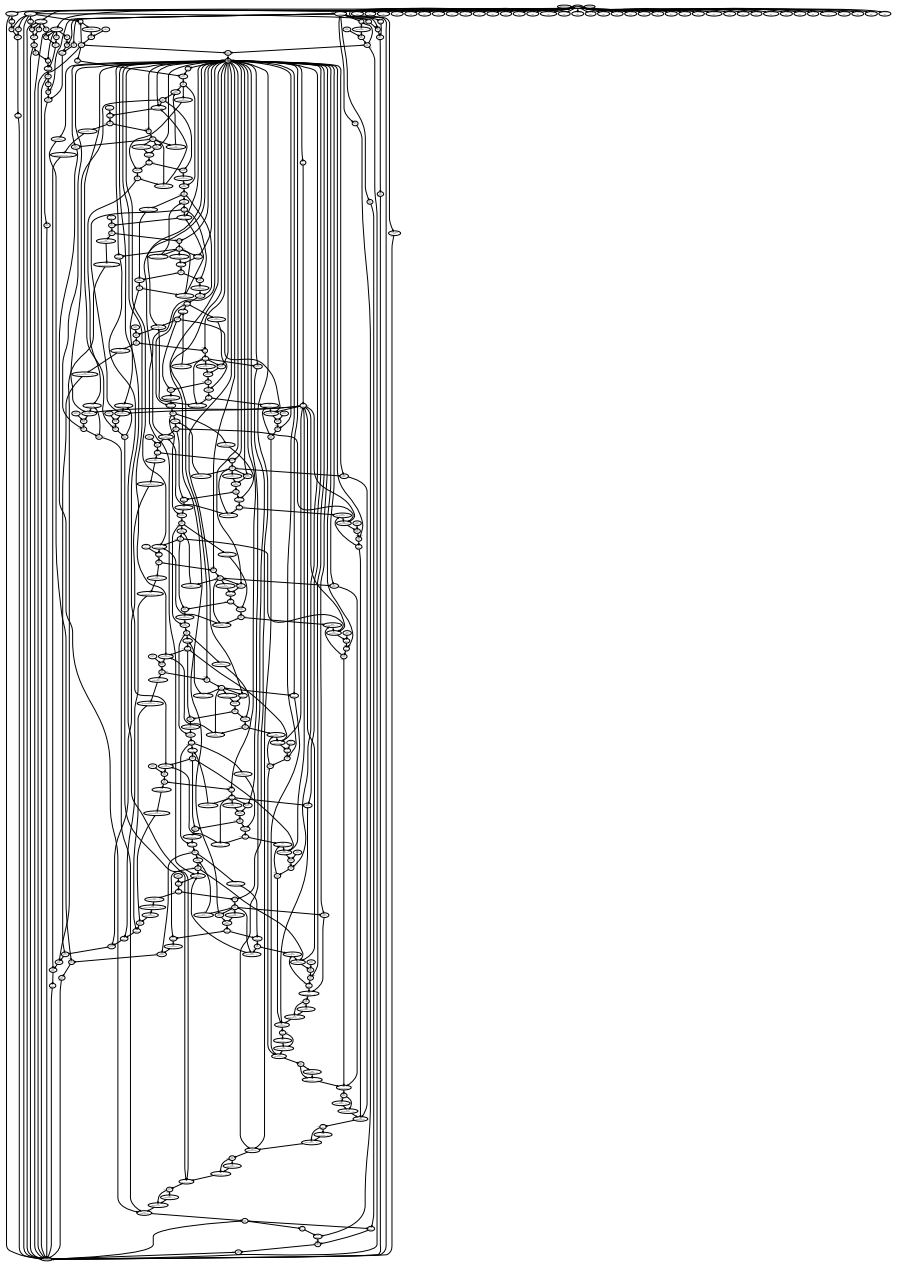
TeSSLa 101: count() macro

```
in event: Events[Int]

def count[A](a: Events[A]) = c where {
    def c: Events[Int] = merge(last(c, a) + 1, 0)
}

def event_count = count(event)

out event_count
```



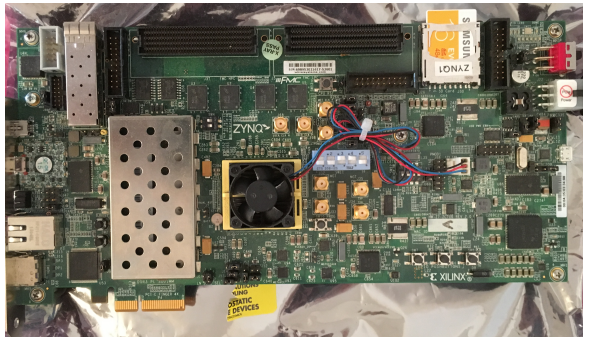
macro count() from TeSSLa stdlib

Unrolled map capacity 8



fixed-capacity map is based on fixed-capacity set by Daniel Thoma

Xilinx Zynq board: a CPU/FPGA hybrid SoC

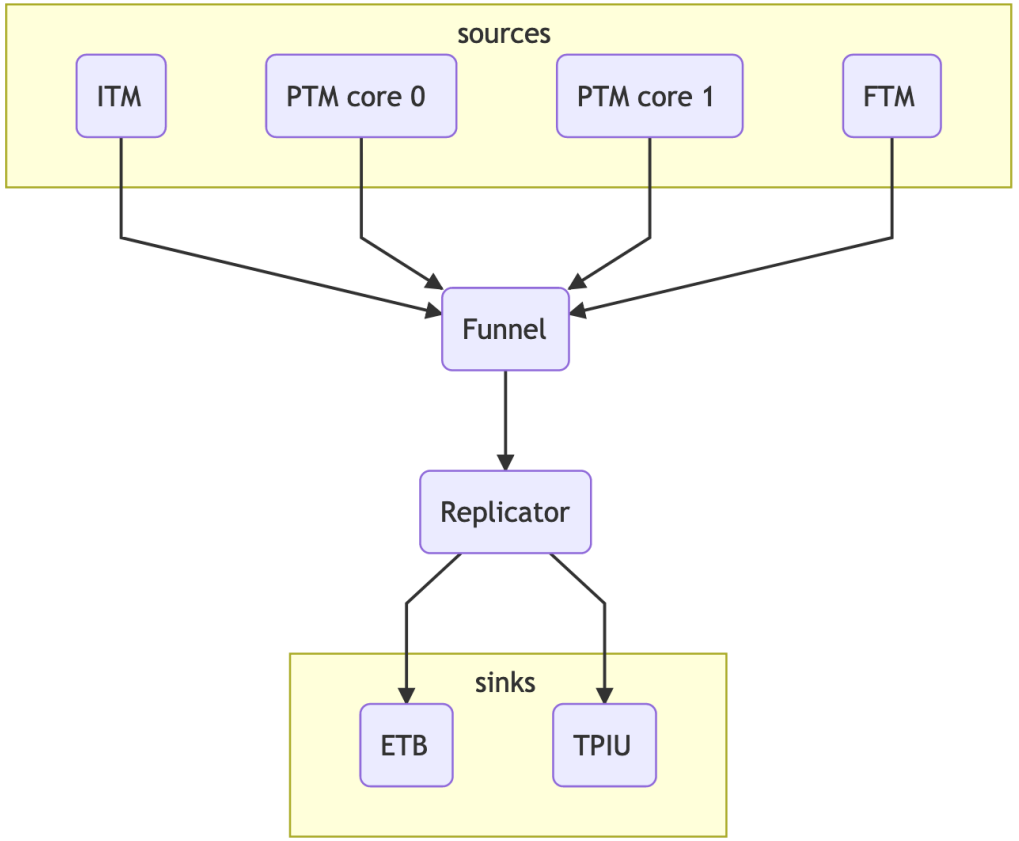


- Zynq-7000 SoC
- 2 ARMv7 cores 1 GHz
- FPGA
- 1 GB RAM each side

- CoreSight (CS) trace available at TPIU interface on FPGA

- Xilinx Linux kernel 4.4 with patch (processID in CONTEXTIDR)
- Ubuntu 18.04 LTS

ARM CoreSight Infrastructure on Zynq-7000 SoC



Required bandwidth

250-300 MB/s (2 cores)

CoreSight Infrastructure on Cavium/Marvell ThunderX

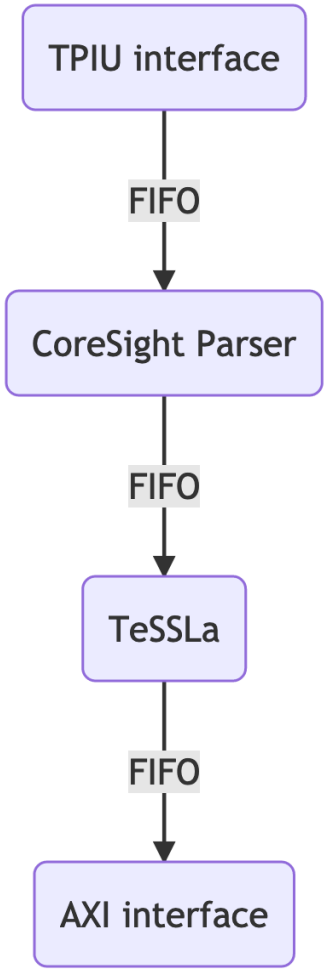
ENZIAN

- 48 cores, 2.5 GHz
- [currently potentially confidential information removed]

Required bandwidth

300-350 MB/s per core

Data flow: CS trace to event stream



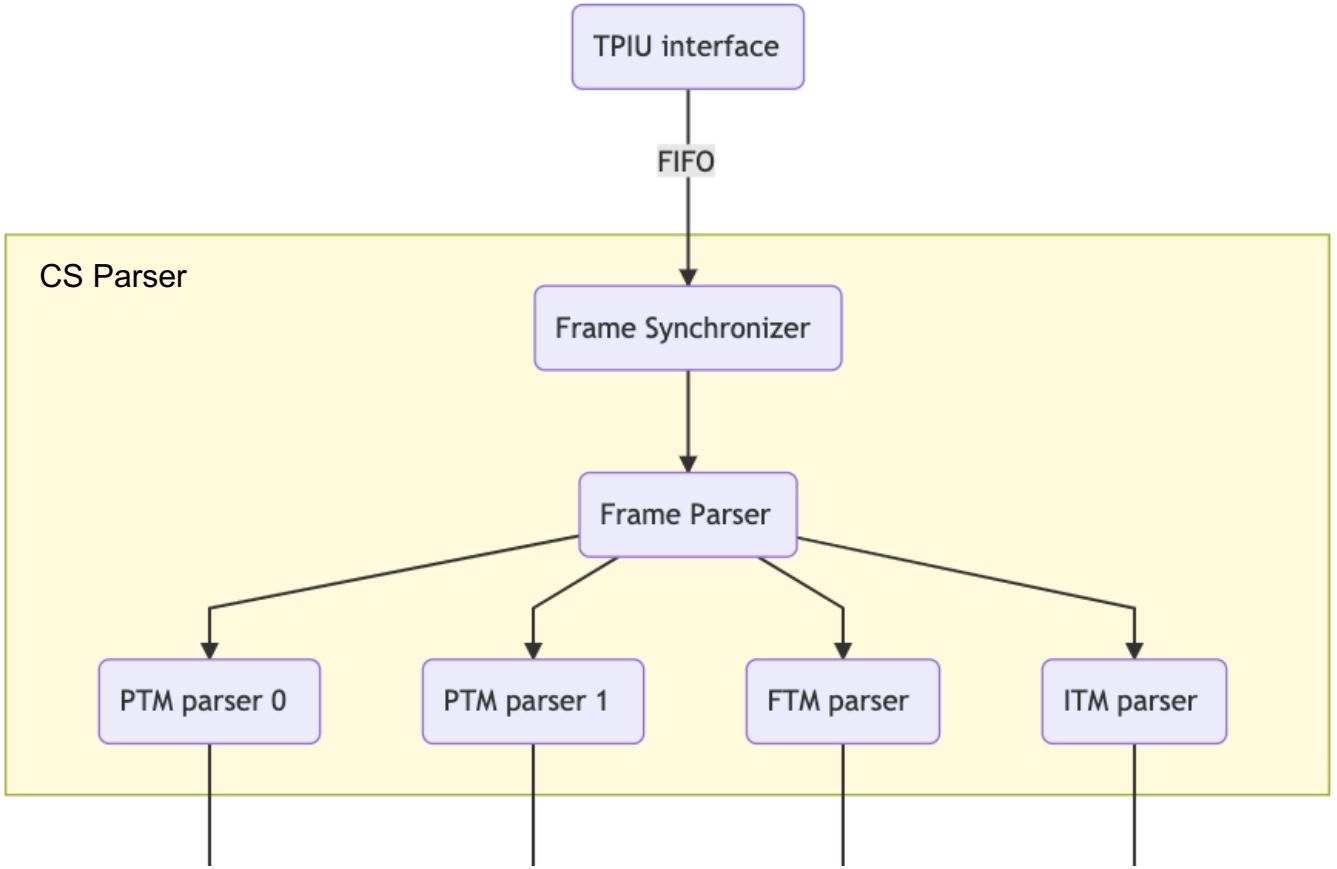
clock 200 MHz; bandwidth **400 MB/s**

clock 125 MHz; bandwidth **500 MB/s per parser**
output: **up to 125 M events/s**

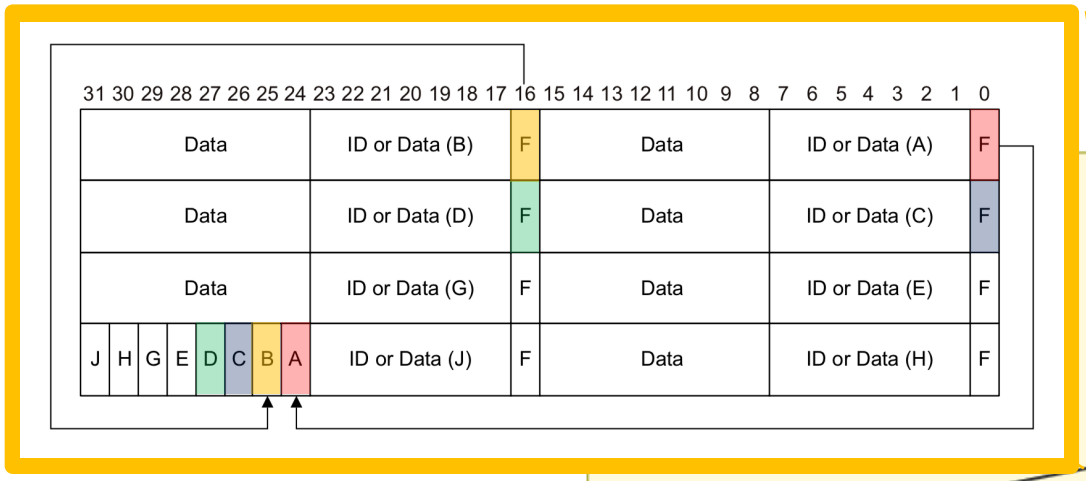
clock 50 MHz; in **25 M events/s per adapter**
out **25-50 M/events/s**

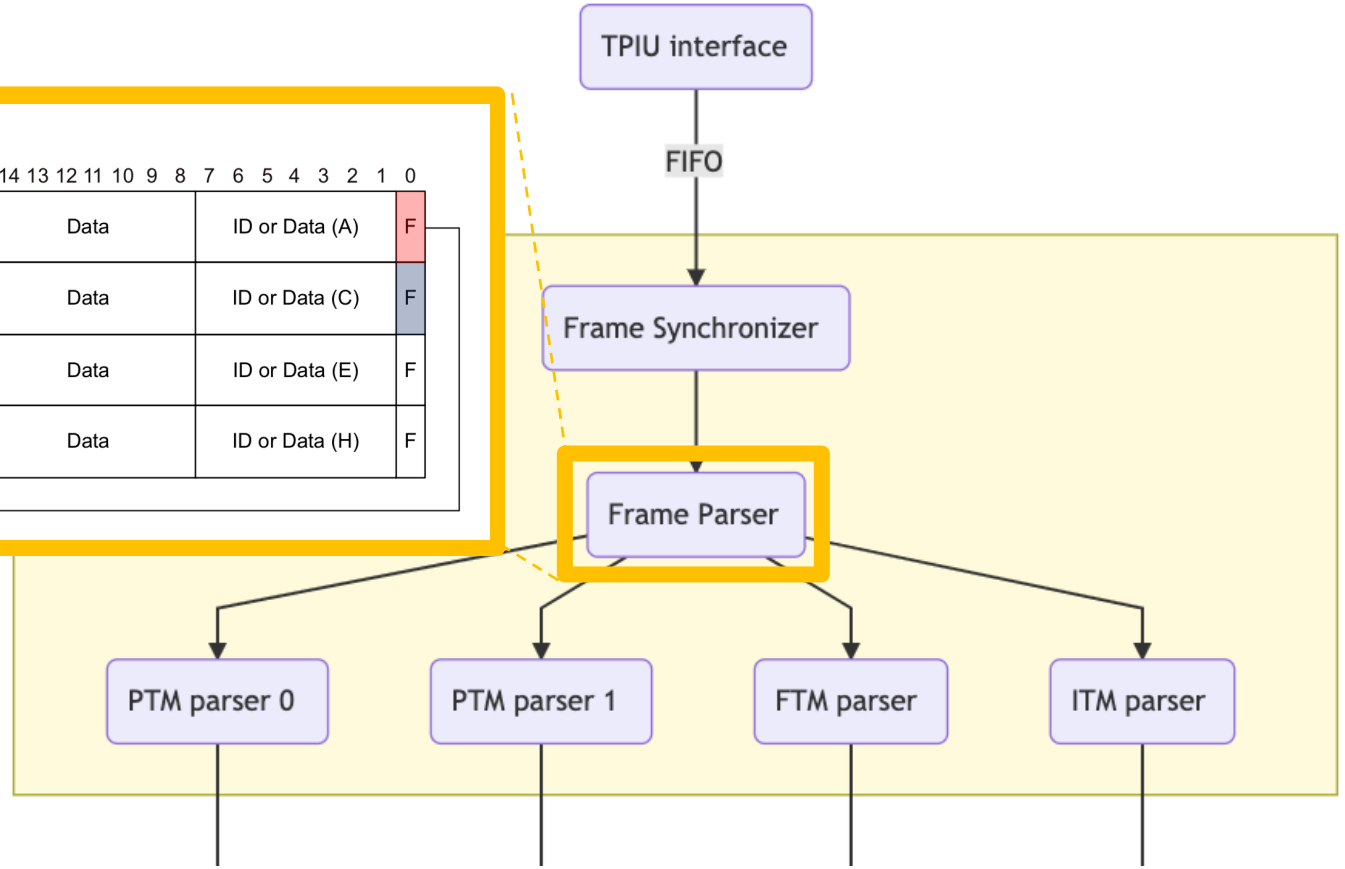
```
Event tuple: (timestamp, data value)
```

Data flow


TPIU interface
FIFO
CS Parser
Frame Synchronizer
Frame Parser
PTM parser 0
PTM parser 1
FTM parser
ITM parser

Data flow

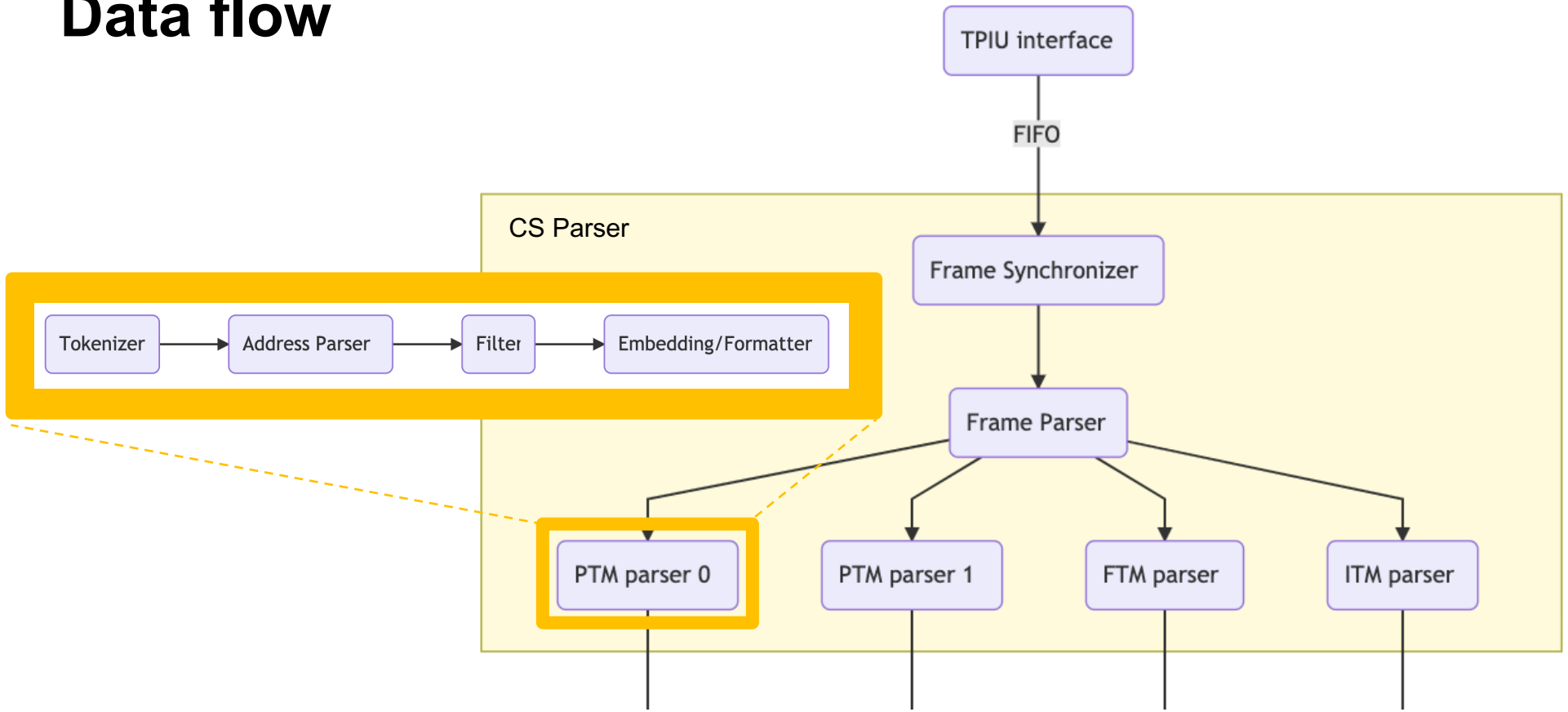




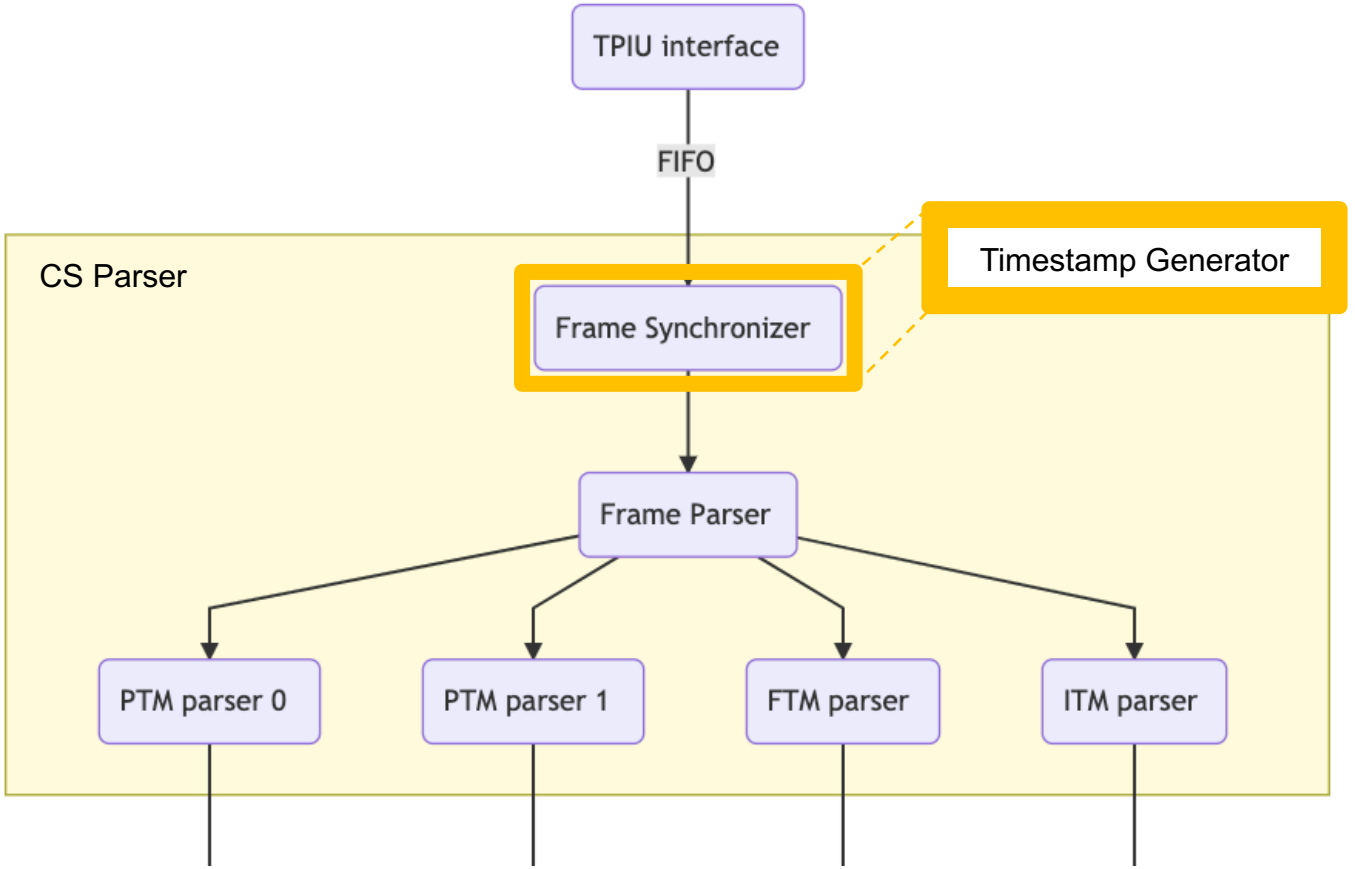
ID	Value
Core 0	0x10
Core 1	0x11
ITM	0x6F
FTM	0x70 to 0x7F

CS frame format from ARM CoreSight Architecture Specification. reference IHI 0029E, ARM Ltd 2017

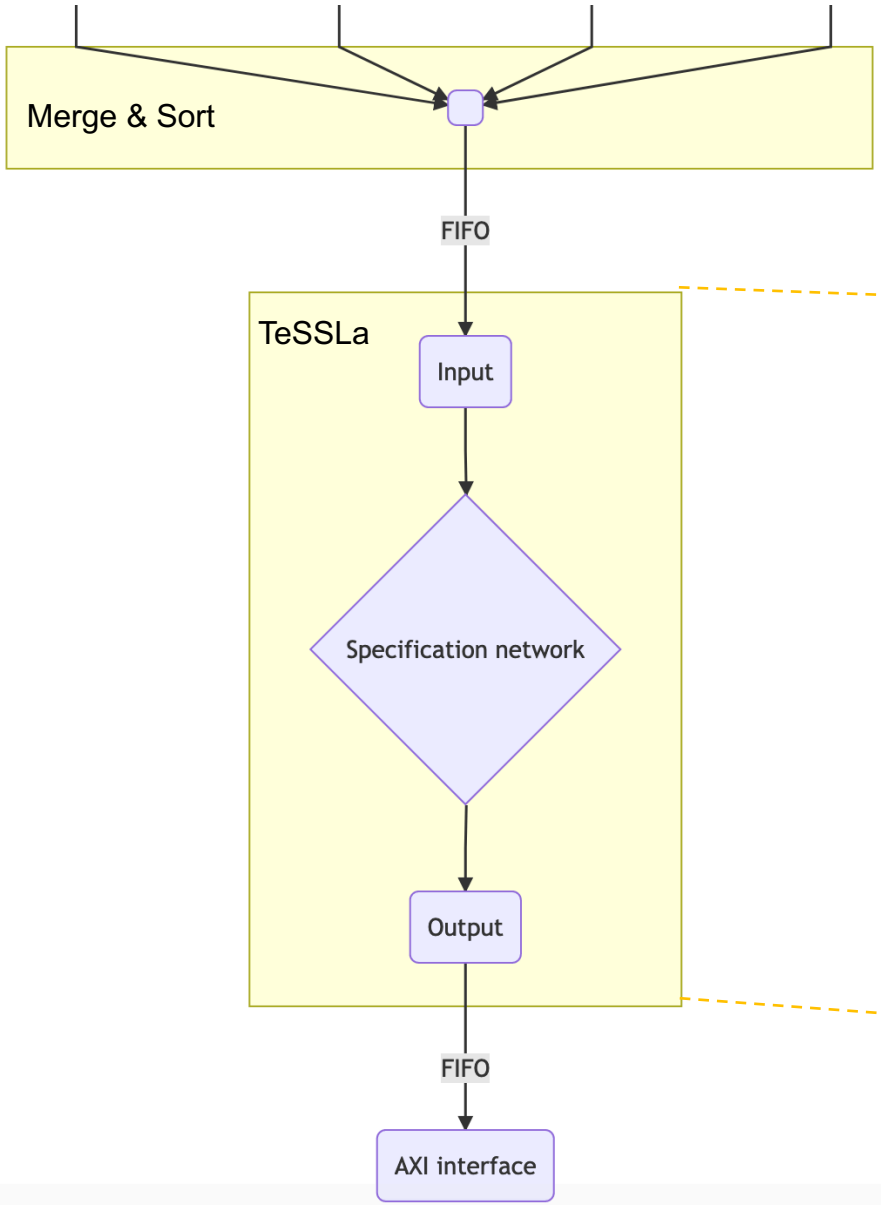
Data flow


TPIU interface
FIFO
CS Parser
Frame Synchronizer
Tokenizer
Address Parser
Filter
Embedding/Formatter
Frame Parser
PTM parser 0
PTM parser 1
FTM parser
ITM parser

Data flow


TPIU interface
FIFO
CS Parser
Timestamp Generator
Frame Synchronizer
Frame Parser
PTM parser 0
PTM parser 1
FTM parser
ITM parser

Data flow



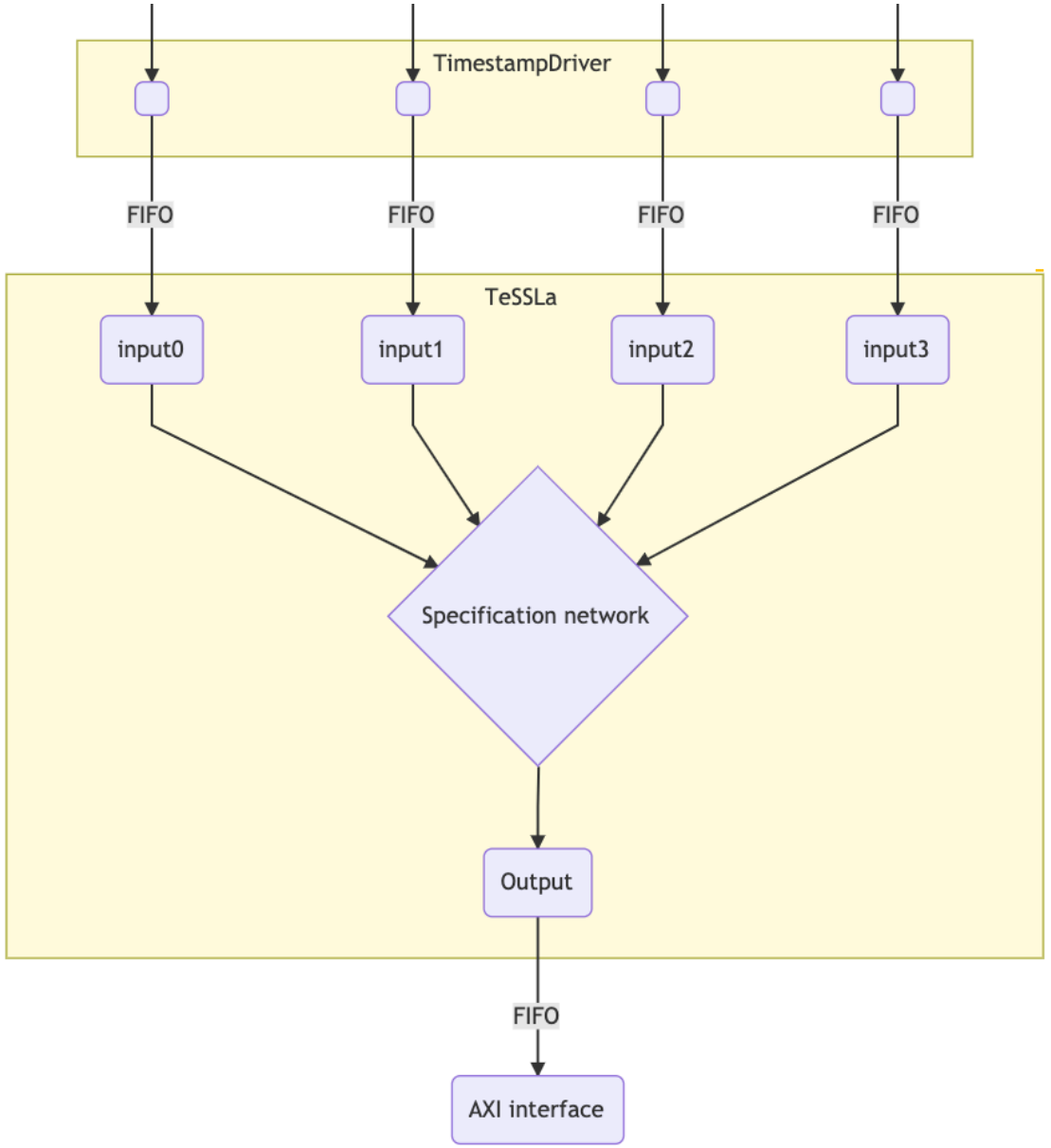
TeSSLa to Verilog prototype compiler

Malte Schmitz
Daniel Thoma

Institut für Softwaretechnik und Programmiersprachen

Universität zu Lübeck, Germany.

Data flow



TeSSLa to Verilog prototype compiler

Malte Schmitz
Daniel Thoma

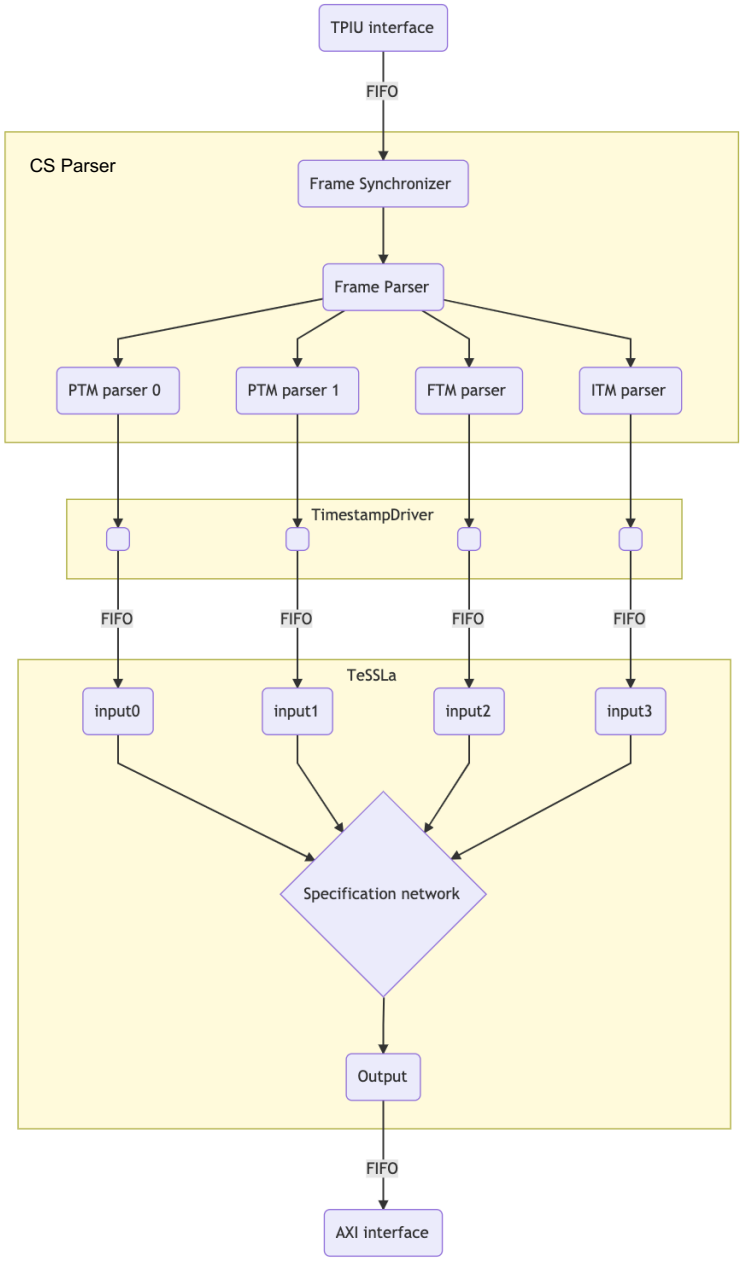
Institut für Softwaretechnik und Programmiersprachen

Universität zu Lübeck, Germany.

Key modifications:

1) InputAdapter → MultiInputAdapter
2) OutputFilterAdapter
3) Various detail improvements and fixes

Data flow


TPIU interface
FIFO
CS Parser
Frame Synchronizer
Frame Parser
PTM parser 0
PTM parser 1
FTM parser
ITM parser
TimestampDriver
FIFO
FIFO
FIFO
FIFO
TeSSLa
input0
input1
input2
input3
Specification network
Output
FIFO
AXI interface

Evaluation & Benchmarking

- PTM: Correct addresses and ContextID
- ITM/FTM: correct values

- Program trace reference: raw CS traces parsed by OpenCSD parser

Evaluation & Benchmarking

- PTM: Correct addresses and ContextID
- ITM/FTM: correct values

- Instrumentation: LD_PRELOAD of glibc functions; ITM stimuli

Table 6.2: Summary: average instrumentation overhead. SD < 0.005 μs for all averages; n=3. O:D ratio = Overhead:Direct function call ratio.

Functions	Direct [μs]	Instrumented [μs]	Overhead [μs]	O:D ratio
malloc/free	0.209	0.593	0.384	1.8

Evaluation & Benchmarking

- PTM: Correct addresses and ContextID
- ITM/FTM: correct values

- Instrumentation: LD_PRELOAD of glibc functions; ITM stimuli

Table 6.2: Summary: average instrumentation overhead. SD < 0.005 μs for all averages; n=3. O:D ratio = Overhead:Direct function call ratio.

Functions	Direct [μs]	Instrumented [μs]	Overhead [μs]	O:D ratio
malloc/free	0.209	0.593	0.384	1.8

compared with Wahab et al. 2018: syscall and CONTEXTIDR ≥ 0.43 on Zynq board

Instrumentation with syscall and CONTEXTIDR (Wahab et al. 2018, ref 31 in thesis): 30 µs per call on Zed Board with Linux ≥ 4.9; on our board: raw syscall 0.431 µs, getpid() 0.490 µs, (SD < 0.001 µs, n=4)

Evaluation & Benchmarking

- PTM: Correct addresses and ContextID
- ITM/FTM: correct values

- Instrumentation: LD_PRELOAD of glibc functions; ITM stimuli

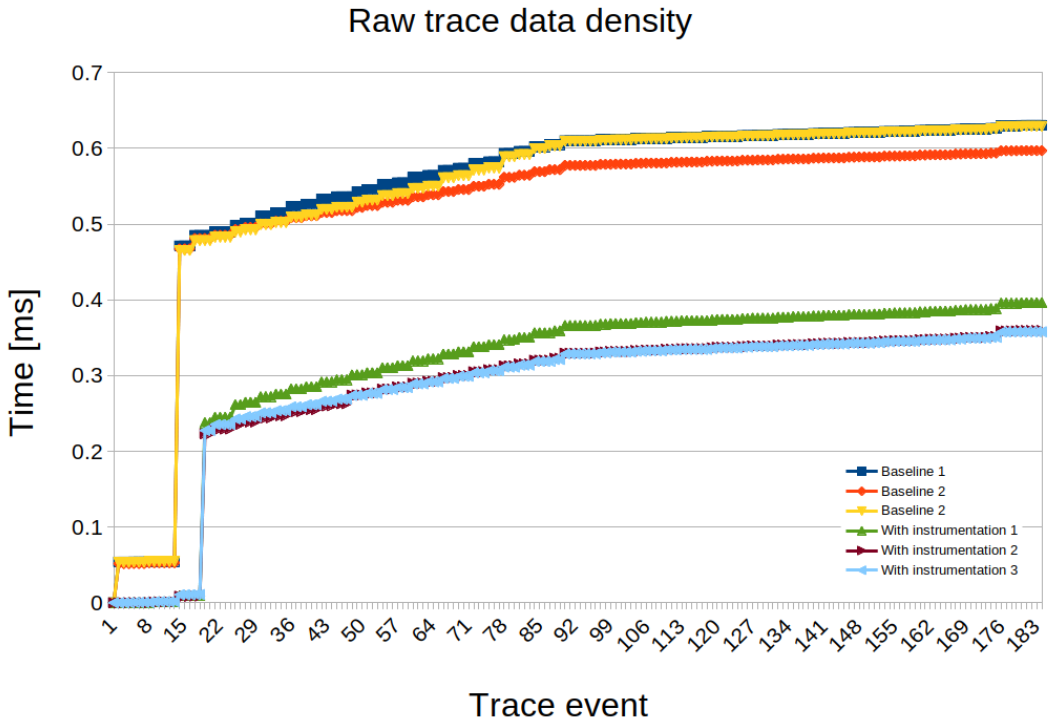
Table 6.2: Summary: average instrumentation overhead. SD < 0.005 μs for all averages; n=3. O:D ratio = Overhead:Direct function call ratio.

Functions	Direct [μs]	Instrumented [μs]	Overhead [μs]	O:D ratio
malloc/free	0.209	0.593	0.384	1.8
lock/unlock, no contention	0.077	0.846	0.769	10.0
lock/unlock, some contention	0.120	0.909	0.789	6.6

Instrumentation with syscall and CONTEXTIDR (Wahab et al. 2018, ref 31 in thesis): 30 µs per call on Zed Board with Linux ≥ 4.9; on our board: raw syscall 0.431 µs, getpid() 0.490 µs, (SD < 0.001 µs, n=4)

Evaluation & Benchmarking

- PTM: Correct addresses and ContextID
- ITM/FTM: correct values
- Instrumentation overhead
- High-resolution observation of programs



Evaluation & Benchmarking

Table 6.1: Runtime verification with TeSSLa specifications: overview

Experiment	Scenario	Methods
14	Memory allocation	preload instrumentation, fixed-capacity set
15	Event handler, simple	direct tracking of function calls
16	Event handler, advanced	additionally fixed-capacity maps
17	Critical section and locks	instrumentation, fixed-capacity sets

Specification I: Memory Allocation

- LTL: $\square(x = malloc(\_) \longrightarrow \Diamond free(x))$

Specification I: Memory Allocation

- LTL: $\square(x = malloc(\_) \longrightarrow \Diamond free(x))$

- TeSSLa:
  - fixed-capacity set
  - malloc(): add address
  - free(): remove address
  - set size == 0
  - no overflows

```
-- instrumentation signals: in_malloc and in_free
def key = merge(in_malloc, in_free)
def allocated = lookup(key, time(in_malloc) == time(key), 8)

def n_allocations = allocated._1
def overflow_flag = allocated._2
def overflow = overflow_flag == 0
def never_overflow = none(overflow)

def all_allocations_freed = n_allocations == 0 && never_overflow

def none(p: Events[Bool]) = c where {
  -- no true events
  -- first result before any events
  def c: Events[Bool] = merge(last(c, p) && (!p), true)
}
```

Fixed-capacity set implementation in TeSSLa provided by Daniel Thoma

Specification I: Memory Allocation

Specification satisfied

```
Last observed value:
 :: @0 [in_malloc] = 168328 (0x29188)
 :: @1 [malloc_count] = 8 (0x8)
 :: @2 [in_free] = 139600 (0x22150)
 :: @3 [free_count] = 8 (0x8)
 :: @4 [n_allocations] = 0 (0x0)
 :: @9 [never_overflow] = true
 :: @11 [all_allocations_freed] = true
```

Specification violated: missing free()

```
Last observed value:
 :: @0 [in_malloc] = 168328 (0x29188)
 :: @1 [malloc_count] = 8 (0x8)
 :: @2 [in_free] = 139600 (0x22150)
 :: @3 [free_count] = 7 (0x7)
 :: @4 [n_allocations] = 1 (0x1)
 :: @9 [never_overflow] = true
 :: @11 [all_allocations_freed] = false
```

Specification violated: additional malloc()

```
Last observed value:
 :: @0 [in_malloc] = 139600 (0x22150)
 :: @1 [malloc_count] = 9 (0x9)
 :: @2 [in_free] = 139600 (0x22150)
 :: @3 [free_count] = 8 (0x8)
 :: @4 [n_allocations] = 1 (0x1)
 :: @9 [never_overflow] = true
 :: @11 [all_allocations_freed] = false
```

Specification not satisfied: overflow

```
Last observed value:
 :: @0 [in_malloc] = 184744 (0x2d1a8)
 :: @1 [malloc_count] = 12 (0xc)
 :: @2 [in_free] = 139600 (0x22150)
 :: @3 [free_count] = 12 (0xc)
 :: @4 [n_allocations] = 0 (0x0)
 :: @9 [never_overflow] = false
 :: @11 [all_allocations_freed] = false
```

Original collected data from experiments. Only relevant parts shown.

Specification II: Event handler, simple

- Property: Queueing time ≤ 5 ms for request type A

Specification II: Event handler, simple

- Property: Queueing time ≤ 5 ms for request type A

- TeSSLa:
  - direct detection of function calls
  - time subtraction and comparison
  - helper predicate all()

```
-- adjust addresses to compiled program binary
def request_a  = function_call(0x00010eb0)
def start_a    = function_call(0x00010c3c)
def finished_a = function_call(0x00010c54)

def simple_queueing_time =
  time(start_a) - last(time(request_a), start_a)

-- queueing time <= 5 ms
def p = simple_queueing_time <= 0x989680
def simple_property = all(p)

def function_call(function_address: Int) = c where {
  def a = filter(etm0_addr, etm0_addr == function_address)
  def b = filter(etm1_addr, etm1_addr == function_address)
  def c = merge(a, b)
}
```

Specification II: Event handler, simple

Specification satisfied (3 additional threads)

```
Last observed value:
 :: @3 [simple_queueing_time] = 1.456584 ms (12)
 :: @6 [simple_queueing_time_below_limit] = true
 :: @7 [simple_property] = true


note: observed range in all 3 experiment runs:
      1.1 to 3.4 ms
```

Specification violated (19 additional threads)

```
Last observed value:
 :: @3 [simple_queueing_time] = 11.059488 ms (2)
 :: @6 [simple_queueing_time_below_limit] = false
 :: @7 [simple_property] = false


note: observed range in all 3 experiment runs:
      8.9 to 12.0 ms
```

Original collected data from experiments. Only relevant parts shown. “all_simple_queueing_time_below_limit” renamed to “simple_property” for space reasons.

Specification II: Event handler, advanced

- Property: Queueing time ≤ 5 ms for request type A
- New requests type A may get enqueued *before*
  processing of former request has been completed or even started

Specification II: Event handler, advanced

- Property: Queueing time ≤
- TeSSLa:
  - direct detection of function calls
  - requires tags and maps to track time!
  - time subtraction and comparison
  - helper predicate all()

```
-- adjust addresses to compiled program binary
def request_a = function_call(0x00010eb0)
def start_a = function_call(0x00010c3c)
def finished_a = function_call(0x00010c54)

-- use counts as tags
def request_a_count = count(request_a)
def start_a_count = count(start_a)

def key1 = merge(request_a_count, start_a_count)
def value1 = time(key1)
def store_kv_pair1 = time(request_a_count) == time(key1)
def tracking_map1 = lookup_map(key1, value1, store_kv_pair1, 8)

def queueing_time_with_map = filter(time(start_a) -
  tracking_map1._2, time(tracking_map1._2) == time(start_a))

-- queueing time <= 5 ms
def q = queueing_time_with_map <= 0x989680
def property_with_map = all(q)
```

Specification II: Event handler, advanced

```
Specification satisfied (3 additional threads)

Last observed value:
 :: @14 [property_with_map] = true
```

```
Specification violated (19 additional threads)

Last observed value:
 :: @14 [property_with_map] = false
```

```
Details of one experiment run:

 7.190672 ms (4) :: @8 [request_a_count] = 1 (0x1)
 9.275360 ms (4) :: @8 [request_a_count] = 2 (0x2)
11.387064 ms (4) :: @8 [request_a_count] = 3 (0x3)
13.496808 ms (4) :: @8 [request_a_count] = 4 (0x4)
15.598664 ms (4) :: @8 [request_a_count] = 5 (0x5)
17.697336 ms (4) :: @8 [request_a_count] = 6 (0x6)
18.498992 ms (0) :: @3 [simple_queueing_time] =     0.801648 ms (12) :: @7 [simple_property] = true
                 :: @9 [start_a_count] = 1 (0x1)
                 :: @11 [queueing_time_with_map] = 11.308312 ms (12) :: @14 [property_with_map] = false
```

Original collected data from experiments. Only relevant parts shown. “queueing_time_with_map_below_limit” renamed to “property_with_map” for space reasons.

Specification III: Critical section, protected by lock

- Property: Critical section only executed when lock is acquired
  and only executed once per lock

- Goal: another thread (without acquiring the lock)
  does not access the critical section

Specification III: Critical section, protected by lock

- Property: Critical section only executed when lock is acquired
  and only executed once per lock

- TeSSLa: with current limitations in instrumentation and TeSSLa compiler
  - max. one lock is used
  - lock is held by max. one thread
  - critical section not accessed, if no lock is held
    → A: additional thread does not access critical section while no lock is held
  - critical section only called once per lock
    → B: additional thread does not access critical section while other thread holds lock
  - A ∧ B → all the time: additional thread does not access critical section

Specification III: Critical section, protected by lock

```
-- a) track critical section (cs) --
def critical_section = itm_value13
def cs_count = count(critical_section)
def cs_as_bool = first(true, critical_section)

-- b) set: max. one lock used --
def locks_set = lookup(in_lock_request_addr,true,3)
def locks_count = locks_set._1
def locks_assert = all(locks_count <= 1)

-- c) set: threads holding lock --
def locked_key = merge(in_lock_acquired_tid,
                       in_unlock_tid)
def locked_set = lookup(locked_key,
 time(in_lock_acquired_tid) == time(locked_key), 8)
def locked_count = locked_set._1
def locked_overflow = locked_set._2 == 0
def locked_never_overflow = none(locked_overflow)
def locked_assert = all(locked_count <= 1)

-- d) cs: one lock must be held --
def cs_with_lock = last(locked_count,
                        critical_section) == 1
def cs_assert1 = all(cs_with_lock)
```

```
-- e) cs: only called once per lock --
def cs_allowed = {
  -- simple version for one lock only
  -- method: a small state machine
  def acquired_lock = first(true, in_lock_acquired)
  def released_lock = first(false, in_unlock)
  def cs_started = first(false, critical_section)
  merge(cs_started, merge(released_lock,
                          acquired_lock))
}

def current_cs_allowed = last(cs_allowed,
                              cs_as_bool)
def cs_assert2 = all(current_cs_allowed)

-- f) combine assertions and side-conditions
def cs_assertion = cs_assert1 && cs_assert2 &&
                   locks_assert && locked_assert

-- optional: to have fewer output signals
def property = filter(cs_assertion, cs_as_bool)
```

Specification III: Critical section, protected by lock

Specification satisfied

```
Last observed value:
 :: @0 [cs_count] = 1024 (0x400)
 :: @1 [property] = true
```

Specification violated

```
Last observed value:
 :: @0 [cs_count] = 1025 (0x401)
 :: @1 [property] = false
```

Original collected data from experiments. Only relevant parts shown.

Specification III: Critical section, protected by lock

Specification satisfied

```
Last observed value:
 :: @0 [cs_count] = 1024 (0x400)
 :: @1 [property] = true
```

Specification violated

```
Last observed value:
 :: @0 [cs_count] = 1025 (0x401)
 :: @1 [property] = false
```

Type 1: no lock held

```
11.500008 ms (0) :: @1 [property] = true :: @4 [acquired_count] = 258 (0x102)
11.501304 ms (0) :: @0 [cs_count] = 258 (0x102) :: @1 [property] = true
11.501304 ms (2) :: @1 [property] = true :: @5 [unlock_count] = 258 (0x102)
11.502016 ms (0) :: @0 [cs_count] = 259 (0x103) :: @1 [property] = false
```

Type 2: while lock is held

```
10.708064 ms (0) :: @1 [property] = true :: @4 [acquired_count] = 257 (0x101)
10.709192 ms (0) :: @0 [cs_count] = 257 (0x101) :: @1 [property] = true :: @12 [current_cs_allowed] = true
10.709192 ms (2) :: @0 [cs_count] = 258 (0x102) :: @1 [property] = false :: @12 [current_cs_allowed] = false
10.709992 ms (0) :: @1 [property] = false :: @5 [unlock_count] = 257 (0x101)
```

Original collected data from experiments. Only relevant parts shown.

Upgrade to Enzian

ENZIAN

- Documentation of my design for Enzian
  - in thesis
  - README in code repository
  - some source code prepared
  - several critical problems have been solved during this project
- New TeSSLa to Verilog Compiler
  - MSc thesis ongoing in Lübeck
  - incorporates insights and suggestions of this project
  - should remove several limitations of the prototype compiler
  - will have extended stdlib

Thank you!

- Prof. Timothy Roscoe and Dr. David Cock
- Malte Schmitz and Daniel Thoma
  University of Lübeck, Germany
- Barrelfish and Enzian teams of the Systems Group

ENZIAN

Logos from public web sites

Bandwidths on Zynq Board

Table 4.1: FPGA modules: clock frequencies and associated bandwidths

Module	Frequency	Info	Bandwidth
TPIU	200 MHz	4 B / 2 cycles	400 MB/s
CS parser in each parser out	125 MHz	4 B / cycle	500 MB/s up to 125 M events/s
TeSSLa in TeSSLa out	50 MHz	one input for each parser one 64 bit output	25 M events/s per input 25-50 M events/s, 400 MB/s
output cache FIFO	200 MHz	32 bit width (AXI), 512 KiB	800 MB/s
to SD card	–	online mode (section 4.3)	$\leq$ SD card write speed

Helper predicates

Listing B.1: TeSSLa: helper predicates

```
def all(p: Events[Bool]) = c where {
  -- all events must be true
  -- first result with first event
  def c: Events[Bool] = merge(last(c, p), true) && p
}

def none(p: Events[Bool]) = c where {
  -- no true events
  -- first result before any events
  def c: Events[Bool] = merge(last(c, p) && (!p), true)
}

def some(p: Events[Bool]) = c where {
  -- at least one event must be true
  -- first result with first event
  def c: Events[Bool] = merge(last(c, p), false) || p
}

def some_with_init(p: Events[Bool]) = c where {
  -- at least one event must be true
  -- first result before any events,
  -- i.e. captures the case that no event was received at all
  def c: Events[Bool] = merge(last(c, p) || p, false)
}
```

Fixed-capacity set by Daniel Thoma

Listing B.3: TeSSLa: fixed-capacity set

```
-- fixed-capacity set
-- TeSSLa macro provided by Daniel Thoma, University of Luebeck,
-- Germany
-- max capacity 15 with current prototype compiler

def prev[T](events: Events[T]) = last(events, events)

def lookup(key: Events[Int], f: Events[Bool], size: Int):
    (Events[Int], Events[Int]) = {

  def l: Events[Int] = last(reg, key)

  def add = f && l == -1
  def found = l != -1 && key == l
  def remove = !f && found

  def reg = merge(if add then key else if remove then -1 else l, -1)

  static if size == 0 then
    (default(nil[Int], 0), filter(const(0, key), f))
  else {
    def result = lookup(key, f && !(add || found), size - 1)
    (result._1 + if reg != - 1 then 1 else 0, result._2)
  }
}
```

Fixed-capacity map

Listing B.4: TeSSLa: fixed-capacity map

```
-- fixed-capacity map
-- an extension of the fixed-capacity set macro
-- max capacity 8 with current prototype compiler
-- limitation: keys and values must be > 0 at the moment
-- this macro is used for advanced TeSSLa examples
-- with event_handler test framework
--
-- input: if f == true:  add key/value pair to map
--                false: lookup & removal of key
-- returns ._1: current map size
--         ._2: if f == true:  always returns -2
--                 false: returns value if key was found, 0 otherwise
--         ._3: 0 if overflow, nothing otherwise
-- version 2019-07-25

def lookup_map(key: Events[Int], value: Events[Int],
    f: Events[Bool], capacity: Int):
    (Events[Int], Events[Int], Events[Int]) = {

  def lk: Events[Int] = last(regk, key)
  def lv: Events[Int] = last(regv, key)

  def add = f && lk == -1
  def found = lk != -1 && key == lk
  def remove = !f && found

  def regk = merge(if add then key else if remove then -1 else lk, -1)
  def regv = merge(if add then value else if remove then -1 else lv, -1)

  static if capacity == 0 then
    (default(nil[Int], 0), default(nil[Int], 0),
     filter(const(0, key), f))
  else {
    def result = lookup_map(key, value, f && !(add || found),
                            capacity - 1)
    (result._1 + if regk != -1 then 1 else 0,
      if lk == key then lv else
          if result._2 > 0 then result._2 else -2,
      result._3)
  }
}
```

ETH zürich

Runtime Verification with TeSSLa on Enzian

MSc Thesis Pirmin Schmid

Supervised by Prof. Timothy Roscoe and Dr. David Cock